1.Introduction

serverless的一些缺陷：

- 对语言的限定严重
- 一些外部依赖很难引入

foster 促进

tailor 专门定制

以AWS Lambda为例，这些serverless服务实际上是为所要运行的一套function集合提供了一个container环境。

hinder 阻碍

考虑到docker的便捷程度，可以很容易地将AWS Lambda上的functions做成Image导出，从而方便快捷部署。

因此本文引入SCAR的概念：**Serverless Container-aware ARchitectur**

> 在serverless平台上透明地运行容器

优势如下：

- 引入容器可以更方便地解决“定制环境问题”，这在之前常规的serverless平台上是无法实现的
- 可以实现跨语言编程
- 定义了一种高并行的事件驱动处理模型

2. Related Work

在[1]中的研究表明，基于serverless平台的微服务，和基于普通云平台的微服务相比，架构总成本可以节省超过70%。

在[2]中提到了现阶段serverless遇到的一些问题及发展趋势

- 代码的异构问题
- 架构的构建问题

而本文提出的架构模式，主要就是为了解决以上两个问题，从而提供一个在serverless平台上通用的应用架构模式

port 移植

seamless [无缝的]

3.SCAR Framework

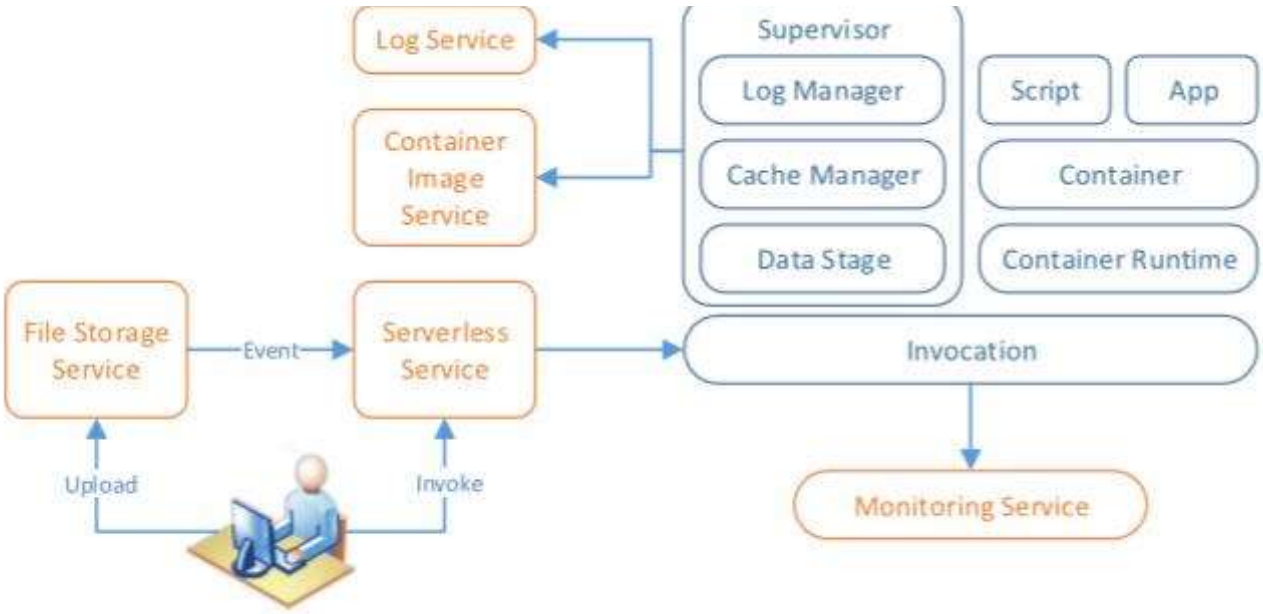


Fig. 1. Architectural approach for supporting container-based applications on serverless platforms.

由云服务提供商提供的包括：

- serverless服务。执行响应事件的function
- 文件存储服务。存储用户上传的文件，并触发serverless服务以便这些文件能够被调用的function进行处理。每次文件上传都会触发一次function调用
- 日志服务。function的执行状态
- 管理服务。function的资源使用情况
- 镜像服务。

每次function调用都涉及到Supervisor的执行。

3.1 相关技术

3.1.1 云服务提供商：AWS

serverless平台选的是AWS Lambda， AWS Lambda相关信息：

- 最大内存使用量为3008MB，CPU性能将与内存大小成正相关
- 最大执行时间为5min
- 只读的文件系统基于Amazon Linux
- 512MB的磁盘空间，不同function可共享
- 默认的单个function最高并发数为1000
- 默认执行环境为Node.js v4,JAVA 8,Python 3.6 and 2.7, .NET core 1.01

文件存储系统选的是Amazon S3，S3可以发布事件通知。比如s3:ObjectCreated可以告诉其他服务创建了一个对象

管理服务使用的是Amazon CloudWatch。

3.1.2 Containers:Docker,DockerHub and udocker

由于serverless平台无法直接安装外部包，Docker就无法进行安装，这里引入udocker——可以在一个无法安装docker的系统上从dockerHub拉取镜像运行容器

3.2 SCAR架构

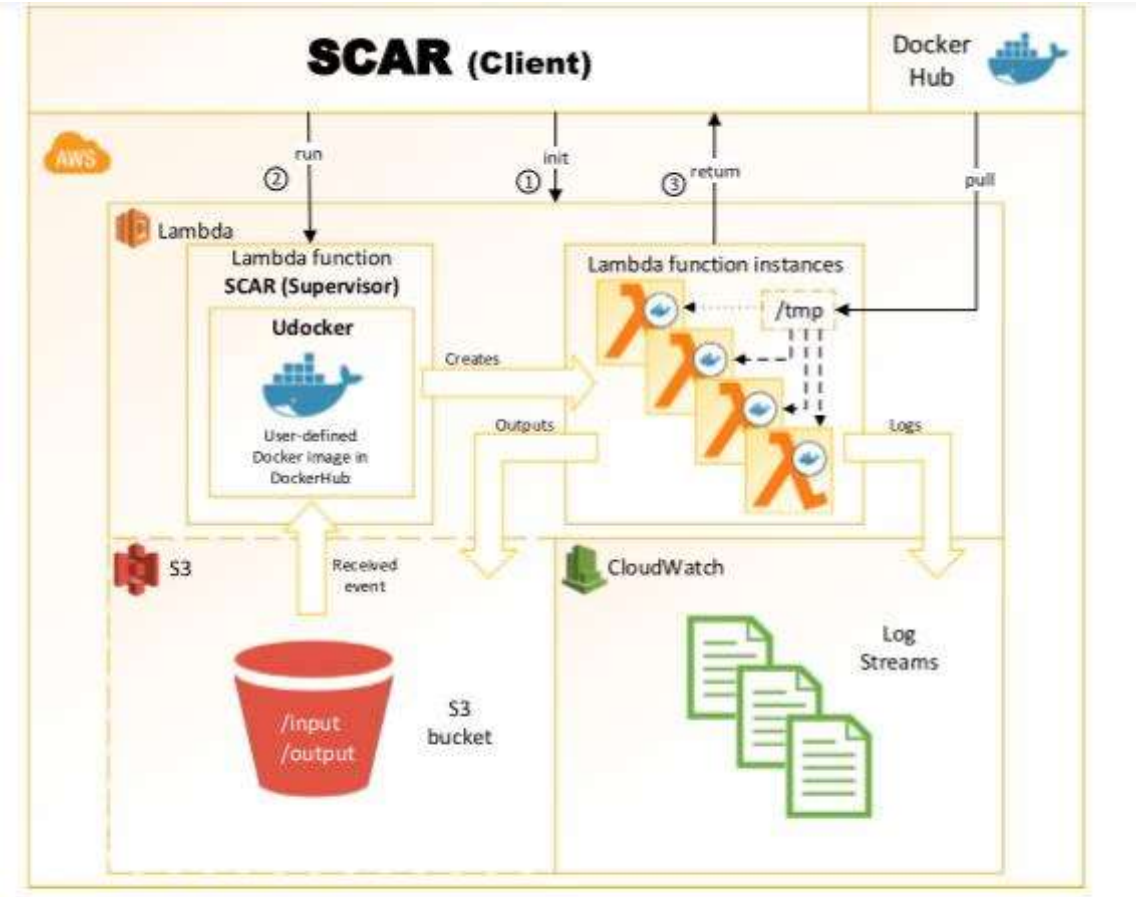


Fig. 2. Architecture of SCAR.

SCAR包括两个部分，允许用户自己定义function，每次调用会将一个docker 镜像实例化，同时启动这个容器中的脚本。

- SCAR 客户端。提供一个包含CLI的Python脚本，主要负责验证用户输入信息、创建包依赖、创建一个包含SCAR supervisor的Lambda函数、提供管理Lambda function生命周期的功能（初始化、运行、删除）、管理触发S3->Lambda function的配置文件
- SCAR Supervisor。 Supervisor类似于虚拟机中的Hypervisor，负责管理生成的容器、从Dockers HUb对镜像进行检索、实例化容器、触发其他服务（S3）、为容器传递环境变量

过程

1. 用户从docker Hub选取一个镜像，同时配置（相关资源大小）并启动好一个Lambda实例

2. 用户运行这个Lambda function，然后function运行完之后将会把之前选取的镜像进行实例化并且执行用户在镜像中定义好的脚本。
3. 数据的状态由supervisor自动与s3进行交互。

Cache

第一次调用Lambda函数将会把镜像拉取到/tmp下，而这里相当于做了一个缓存，之后每次调用就不需要重新拉取了。

所以，使用SCAR进行Function 调用的过程很大程度上取决于在缓存中检索Image的能力

3.3 事件驱动型文件处理serverless编程模型

SCAR模型引入了一个编程模型来处理文件，一旦S3有文件需要处理就会触发这个模型进行处理。

文中给出的事例：

> 一个视频转换模型，定义好一个Lambda函数和一个路径，一旦有视频传到该路径，就会将该文件转化为黑白类型。

具体工作流:

1. 用户将视频上传至scar-test s3 bucket
2. 该文件将被设置为容器可获得
3. 脚本对文件进行转化
4. 生产的文件自动存放入s3的输出文件夹下

脚本内容如下：

```
OUTPUT_DIR="/tmp/$REQUEST_ID/output"

echo "SCRIPT: Invoked Video Grayifier. File available in $SCAR_INPUT_FILE"
FILENAME=`basename $SCAR_INPUT_FILE`
OUTPUT_FILE=$OUTPUT_DIR/$FILENAME
echo "SCRIPT: Converting input video file $SCAR_INPUT_FILE to grayscale to output file $OUTPUT_FILE"
ffmpeg -loglevel panic -nostats -i $SCAR_INPUT_FILE -vf format=gray $OUTPUT_FILE < /dev/null
```

可以看到，实际执行转换的编程语句只有最后一行，就是利用FFmpeg 工具实现视频转换。

由于数据的状态都由Supervisor负责，所以用户只需要关心处理数据的过程就可以，同时由以上的事例也可以看出，可以同时执行多个视频转化任务，每个任务单独开一个容器即可。

retrieve 取回，检索

4. 结果分析

这个部分对SCAR进行一个完整的评估，其运行环境仍为AWS Lambda。

- 最大内存为512MB
- 最长执行时间为300s

根据[3]Lambda工作机制说明文档的解释，一旦有一个Lambda函数被调用，就会启动一个容器，同时这个容器会被保持一段时间，以方便复用。当Lambda函数结束之后，会对该容器进行freeze，而一旦需要复用，则进行thaw操作。

而用户用SCAR所实例化的容器是运行在上述容器之中的。

4.1 磁盘空间复用

该项实验目的是证明freeze/thaw机制能够极大的提高访问效率，缩短服务时间。

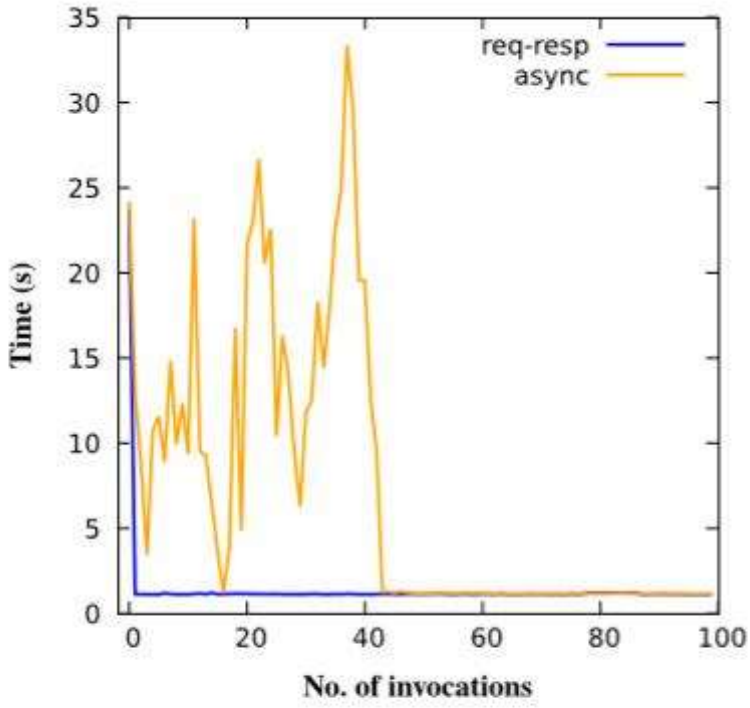
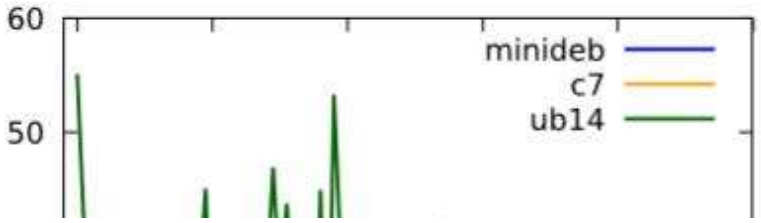


Fig. 4. Average execution time (in seconds) for each invocation type (i.e. request–response and asynchronous).

如图是在SCAR下，AWS Lambda function的两种调用方式，一种是请求—相应式，还有一种是异步式（可以同时并发请求很多）。可以看到请求响应式在第一次调用function以后，之后的每次调用时间都变得非常快，这是因为在freeze/thaw机制下，不用每次都对镜像进行实例化；而异步式调用在40次左右之后的调用，调用时间才变快，这是因为前面的几十次并发请求还没来得及对镜像做缓存处理。



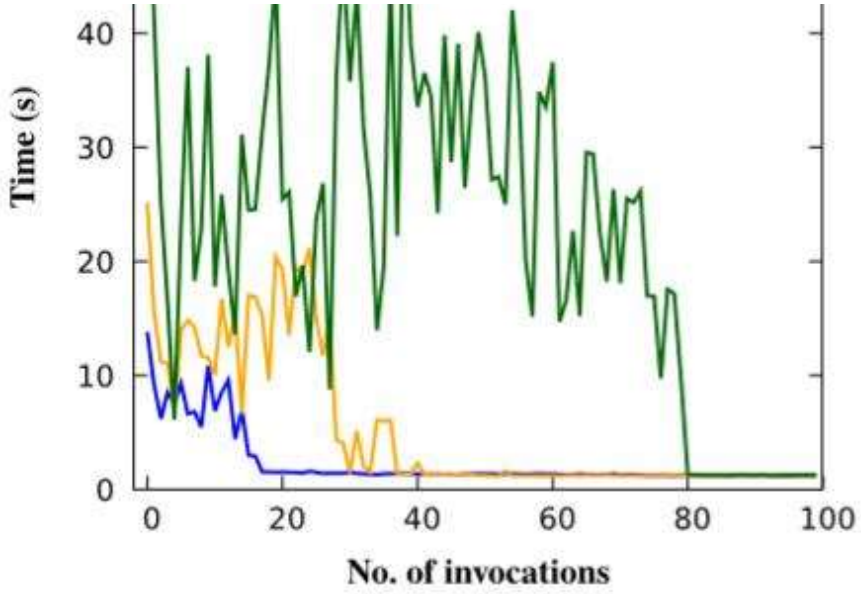


Fig. 5. Average execution time (in seconds) for different container sizes. All the invocations are asynchronous.

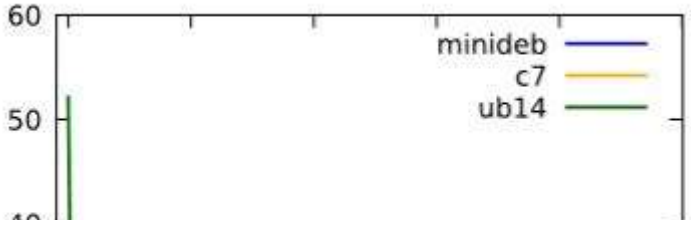
这幅图是比较不同镜像大小时，调用lambda function的时间，其中minideb大小为20MB，c7大小为70MB，ub14为140MB，三种镜像都是基于异步式的访问模式。可以看到，镜像越小，越快进入“cache”模式。

通过上面这两个实验，可以发现：

- **请求响应式的单次访问很快（在第一次之后），但是由于不能并行相应请求，因此所有请求串行的完成时长代价还是很高的**
- **而异步式虽然可以并发处理请求，但由于“缓存时长取决于镜像大小”，所以导致开始时的很多调用单次时间都很长**

基于以上两点，本文提出一种预热的方法，即第一次先用请求响应式，将镜像加载完成后，再采用异步式。

采用预热式之后的实验结果如下：



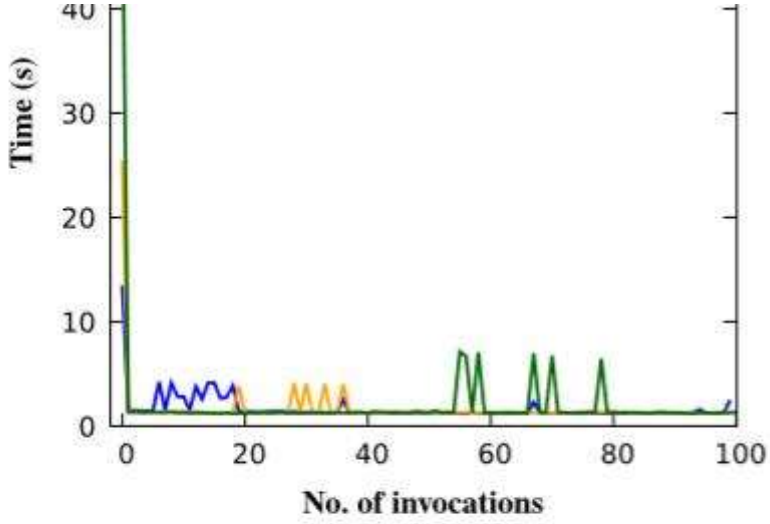


Fig. 6. Average execution time (in seconds) for different container sizes. The first invocation type is request–response and subsequent are asynchronous.

可以看到，消除了前面产生的大部分波动。（中间的波动应该是thaw耗时）

4.2 海量图片处理

问题描述：采用SCAR进行图像中的物体识别，采用的识别工具为Darknet，Lambda内存分配大小为1024MB。

实验相关：

- 一千张左右的图，总大小大约500MB，存放在S3
- 使用的镜像已经安装了所有需要的依赖
- 容器中的脚本会直接对输入的图片进行分类

实验总结：在两分钟内，在没有做任何部署的情况下，完成了一千张图的下载、处理以及上传

结论

1. 没有SCAR，用户几乎不能使用类似于Darknet这种开源库----container的好处
2. 用户不需要去考虑计算的部署、计算流的设计，serverless完全支持异步----serverless的好处
3. SCAR只需要用户在容器内写好一个shell脚本即可。
4. pat-per-use，节省了大量开支----降低了常规容器架构的经济开销

reference

[1]. Cost comparison of running web applications in the cloud using monolithic, microservice, and AWS Lambda architectures

[2]. Serverless computing: Current trends and open problems